AUTOMATING EXPLORATORY TESTING

Presented by Andreas Ulrich, Siemens AG
Exploratory Testing

• James Bach: “Testing is an exploratory process. It’s not just sometimes exploratory; it is inherently exploratory.”

• **Exploratory Testing** is defined as simultaneous
  • Learning,
  • Test design and
  • Test execution.
Exploratory testing $\rightarrow$ informal, decided moment by moment by the tester*

Scripted testing $\rightarrow$ formal, determined by someone else or at some earlier time*

* James Bach, https://www.satisfice.com/exploratory-testing
Adaptive Testing

- Testing of self-contained software component

- Only the component API is accessed → black-box test

- Scenario tests
  - Structured sequence of predefined test commands

- Tester reacts to SUT responses at runtime
  - Selection of next test command according to an overall goal

Adaptive testing approach

```java
Given: Set of Test Commands
var tcmd = Reset();
while(true)
{
    tcmd.Invoke();
    tcmd = SelectNext();
}
```
Test Commands

• A test command = SUT interaction & local state update
  Follows the “4-As” pattern; extension of “3-As” for unit testing
  • **Arrange**: prepare input parameters for SUT call
  • **Act**: perform SUT call
  • **Assert**: validate correctness of returned data
  • **Adapt**: update local state in tester

• A test command is conditioned
  • A **condition** describes the state that enables the test command
  • Finding the right condition and state representation is a **creative, exploratory act**

**Example: Stack**

```java
[Condition(true)] TPush()
{ sut.Push(x);
  assert(sut.Length, i+1); i++;
}

[Condition(i > 0)] TPeek()
{ sut.Peek();
  assert(sut.Length, i) }

[Condition(i > 0)] TPop()
{ x = sut.Pop();
  assert(sut.Length, i-1); i--; }
```
Coverage as Overall Goal

- When is a test command covered?
  - When it was executed.

- When is the set of test commands covered?
  - Syntactical coverage over test command definitions
    - Counting test commands, pairs of test commands etc.
  - Semantical coverage over states reached in test execution
    - Counting states
Reinforcement Learning

• Elements in reinforcement learning
  • **State**: Set of test commands enabled in current location
  • **Action**: Next test command to be executed
  • **Reward function**: Maximise a chosen coverage criterion

• Selection of next test command is a **stochastic process**, which results into SUT **exploration**

• **Issue**: Recognise states of SUT, which is black-box
  • Use approximations
  • Learn states from performed test sequences

(figure source: wikipedia.org, adapted)
What Difference Does It Make – Example 1: Deep State Failures

- **5 adaptive tests** detected a failure that 3453 unit tests and 1084 hard-coded integration tests were unable to find!
- Adaptive tests remain effective in finding failures over the entire development cycle

<table>
<thead>
<tr>
<th>Test Type</th>
<th>Date/Time</th>
<th>Duration</th>
<th>Average Duration</th>
<th>Passed</th>
<th>Failed Fixtures</th>
<th>Failed</th>
<th>Run</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptive Tests</td>
<td>2017-09-20</td>
<td>8m 3.523s</td>
<td>1m 20s</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>Unit Tests</td>
<td>2017-09-20</td>
<td>1h 17m</td>
<td>1h 17m</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3453</td>
</tr>
<tr>
<td>Integration Tests</td>
<td>2017-09-20</td>
<td>5h 26m</td>
<td>5h 26m</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1084</td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>2h 15m</td>
<td>2h 30m</td>
<td>4530</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>4512</td>
</tr>
</tbody>
</table>
What Difference Does It Make – Example 2: Performance Degradation

• Tracking of the execution time of test commands
  • Within the same test run → Reliability tests
  • Over multiple test runs over time → Regression tests

• Expectation on execution times
  • Constant
  • Proportional
  • Learned

Repeated execution times (in µsec) of the same test command in a single test run.
Conclusions

• Exploratory testing doesn’t need to stop with scripted tests

• Adaptive testing preserves the exploration capability during runtime

• Reinforcement learning helps find optimal solutions to a given coverage goal utilising:
  • Exploration of potentially new SUT behaviour
  • Exploitation of learned knowledge